FGB: A Graphical and Haptic User Interface For Creating Graphical, Haptic User Interfaces

Tom Anderson
Sandia National Laboratories
tgander@sandia.gov

Arthurine Breckenridge
Sandia National Laboratories
arbreck@sandia.gov

George Davidson
Sandia National Laboratories
gsdavid@sandia.gov

Abstract

The emerging field of haptics represents a fundamental change in human-computer interaction (HCI), and presents solutions to problems that are difficult or impossible to solve with a two-dimensional, mouse-based interface. To take advantage of the potential of haptics, however, innovative interaction techniques and programming environments are needed. This paper describes FGB (FLIGHT GHUI Builder), a programming tool that can be used to create an application specific graphical and haptic user interface (GHUI). FGB is itself a graphical and haptic user interface with which a programmer can intuitively create and manipulate components of a GHUI in real time in a graphical environment through the use of a haptic device. The programmer can create a GHUI without writing any programming code. After a user interface is created, FGB writes the appropriate programming code to a file, using the FLIGHT API, to recreate what the programmer created in the FGB interface. FGB saves programming time and increases productivity because a programmer can see the end result as it is created, and FGB does much of the programming itself. Interestingly, as FGB was created, it was used to help build itself. The further FGB was in its development, the more easily and quickly it could be used to create additional functionality and improve its own design. As a finished product, FGB can be used to recreate itself in much less time than it originally required, and with much less programming. This paper describes FGB's GHUI components, the techniques used in the interface, how the output code is created, where programming additions and modifications should be placed, and how it can be compared to and integrated with existing API's such as MFC and Visual C++, OpenGL, and GHOST.

I. Introduction

A significant bottleneck in productivity for the use of 3D computer generated environments is the relatively poor interface between the user and the computer. While 3D computer graphics and 3D applications have become much more complicated, interfaces to these 3D environments have remained consistent for decades. To unlock the potential of today's computers, advanced Human-Computer Interface (HCI) techniques are needed. There have been several recent thresholds that allow a new, fundamental shift in the ways that people interact with computers. Graphics cards, for example, have reached prices and rendering capabilities where personal computers can effectively utilize 3D graphics. One of the most important thresholds, however, is the ability to use our sense of touch in computer environments, which will be a critical element of future interfaces.

As with any computer technology, both software and hardware must be developed in parallel to make the technology effective. The dramatic improvements in haptics hardware are now being followed by more advanced research in haptics software. This paper describes FGB, a programming tool that can be used to accelerate haptics software development and research. FGB, a graphical and haptic interface itself, can help a programmer quickly create an application specific interface that utilizes 3D graphics, haptics, and 3D (HRTF) sound. It contains a variety of components such as buttons, sliders, text boxes, etc. that can be used to create an interface. These objects, and the

DISCLAIMER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

3D windows that contain them, called control modules, are easily manipulated in the FGB GHUI without any need for programming. A programmer can create an interface in an intuitive way by creating and modifying components using techniques that mimic real life. Objects can be moved directly with all six degrees of freedom, and the programmer can immediately see the end result of the GHUI as it is created. FGB then outputs appropriate code that can be compiled to create the specified GHUI.

A developer working in FGB, for example, can click and drag the cursor to create a 3D window containing objects needed in an application. If a developer desires a button, it can be created and named, and can then be directly positioned and sized within the 3D window on any wall, or even in the air. The button can be set to call a callback when it is pushed in any direction. Any modifications are handled through the FGB GHUI, which has all of the options available for any selected object. These options can be modified directly with the cursor or through text input boxes. The system therefore represents an efficient tool that can be used to quickly create a user interface that a programmer can use as a front-end to haptic applications.

The code that is output uses the FLIGHT Application Programming Interface (API). FLIGHT is a development environment that was created for rapid haptic application development [2]. It is both a user interface and a programming interface. FLIGHT uses a craft metaphor in which a user can navigate the craft through a virtual space, utilize 3D tools, and interact with a 3D control panel and user interface. FGB is a tool that aids in programming within the FLIGHT development environment.

II. GHUI Components

There are several components currently available that can be used in creating a GHUI. These include control modules, buttons, sliders, button panels, text, text input boxes, force callbacks, graphics callbacks, and removal callbacks. The objects and their descriptions are given in Table 1. All of the objects given in Table 1 can be sized, scaled, positioned, and otherwise modified to fit the needs of the application while still maintaining a consistency in developed interfaces.

Control Modules	Control Modules are windows that are analogous to the common 2D windows metaphor, except that they are three-dimensional. They can be moved and scaled at any time by a user. They contain the objects listed below, in any configuration, and are the base element to create a FLIGHT GHUI. The control module walls can be made to disappear while they still contain objects or callbacks, so any of the objects below can appear to be a standalone object. Control Modules can be moved to any domain in an application. For example, they can be used within the control panel (in the craft metaphor) or near the application specific data.
Buttons	Buttons are components that are similar to their real-life counterparts. When they are pushed, a user can feel them click. They can be attached to walls, can be set to move in any direction, and have a callback associated with each movement direction that is enabled.
Sliders	Sliders are also analogous to their real-life counterparts, except that they are more flexible. They can be pushed along any axis and therefore can control three-dimensional variables directly (for example, a 3D slider can control the RGB components of color). Sliders can be set to control any variable and can be automatically adjusted by the system in real time if that variable changes elsewhere in the code.
Button Panels	As their name suggests, button panels are two-dimensional panels of buttons. There is a callback associated with a button panel that receives the number of a button if it is activated and another one that receives the number of a button if it is deactivated. The buttons can be activated and/or deactivated depending on the following button panel modes: CM_BP_TOUCH: While a button is pressed, it is on. Otherwise it is off. CM_BP_TOGGLE: When a button is pressed, it toggles on or off. CM_BP_ ONE: Exactly one button is always active in the button panel. CM_BP_MAX_ONE: At most one button is on, but the active button can be turned off.
Text	This is a text object in the control module. Text (and text input boxes below) can be sized, scaled, colored, and positioned.
Text Input Boxes	This is a text input object in the control module. These objects can be set to return ints, floats, or character strings. They are two-dimensional, however, they can be positioned anywhere in 3D space. To activate them, a user touches the text input box and a flashing cursor appears

	indicating that it is active. The tab key can be used to move between text input boxes.
Force Callbacks	Force callbacks are called during each cycle of the haptics programming loop within FLIGHT. They can be made to be called only when the cursor is in the vicinity of the control module if desired. A control module can have multiple force callbacks in any order.
Graphics Callbacks	Graphics callbacks are called during each cycle of the graphics loop within FLIGHT. They can be made to be called for the control panel or world interaction modes in FLIGHT. A control module can have multiple graphics callbacks in any order.
Removal Callbacks	A removal callback is called when a control module is deleted.

Table 1: GHUI Components.

Objects can be created in any or all domains (i.e. an object can be seen but not felt, or it can exist in the graphics, haptics, and sound domains). All of the properties for the objects can be changed while the application is running. For example, if a button is only applicable in certain situations, it can be programmed to appear or disappear appropriately. There are a number of other components that will be available in the next release of FLIGHT and FGB. Several of these are toolbars, wizard components, and tabs for different uses within a single control module.

III. FGB Techniques

When a programmer starts FGB, the base control module appears. It can be used to create a new control module, modify an existing one, and add components into a control module. When a control module is selected or created, its attributes can be modified directly with the cursor or through text input boxes. A powerful feature of FGB is that it can modify any of FLIGHT's existing control modules. If, for example, there is a control module already built that is similar to one that is desired, it can be highlighted and modified, and then the new programming code for it can be written to a file.

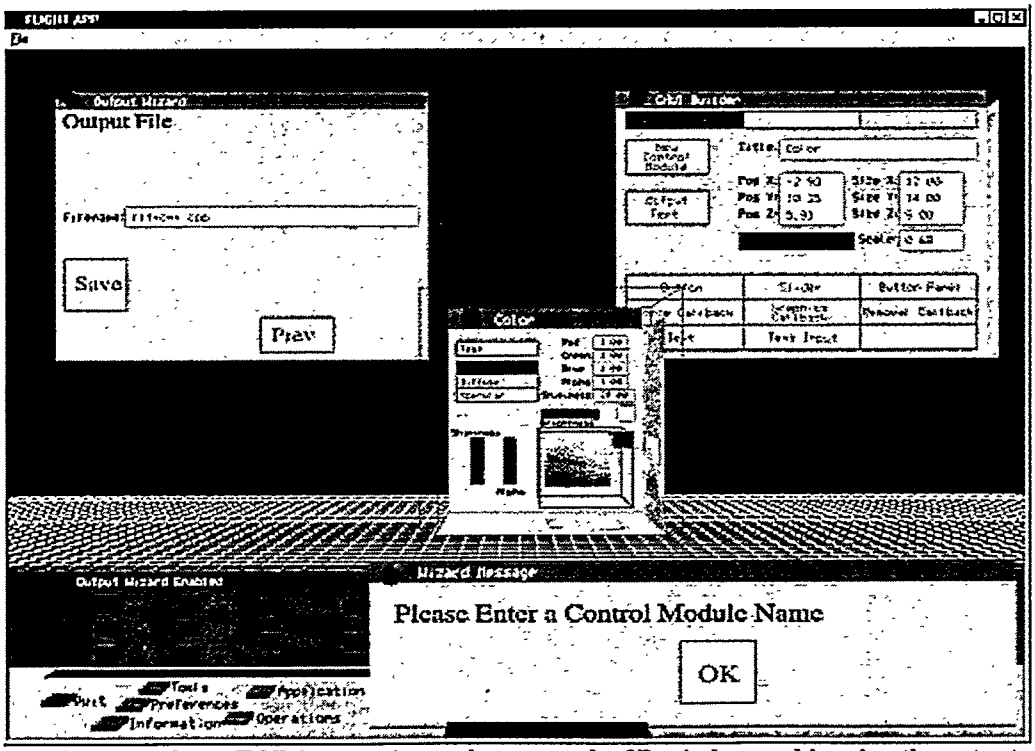

Figure 1: This figure shows FGB in use. A user has created a 3D window and is using the output wizard to output the programming code. The wizard is prompting the user for additional information needed to create the output file.

Through the base control module in FGB, any number of objects can be added to or deleted from a control module. When an object is added or selected, another of FGB's control modules appears to adjust the parameters.

Objects can also be positioned, sized, or scaled directly with the cursor, or can be adjusted through text input boxes. Whenever an object is highlighted, all of its available options are given in the FGB Objects Control Module.

After a control module is created and its objects are in place, a user can generate the code through the output wizard as shown in Figure 1. The output wizard leads a user through an intuitive process of gathering any additional information that is needed to output the code. For example, the output wizard makes suggestions for variable names that the control module code will use, and allows the user the option of changing any of those names or variables. The final step in the output wizard saves the programming code for the active control module. Before the code is saved, the wizard makes sure all the necessary information has been entered and that there are no discrepancies. If there are any errors, an error message appears and the user is taken directly to the step in the wizard where the error occurred. A control module's programming code is output to a modular C++ file, which is ready to be compiled without any further modifications. The file contains the callbacks that were created for any of the GHUI's components, where additional programming code can be added to describe the components' functionality.

After an application specific GHUI is created, it can be incorporated into other applications or API's through FLIGHT's programming interface. FLIGHT has three primary functions that are used to incorporate it into other applications: an initialization callback, a graphics callback, and a haptics callback. The initialization callback sets up memory for process communications and initializes the system. The graphics callback is called each cycle of an OpenGL based graphics loop. At the lowest level, the haptics callback simply receives a device's position and outputs a force based on that position. Because of this simple programming interface for integrating FLIGHT into existing applications, FGB represents a tool that can be used to create interfaces for a variety of applications. For example, FLIGHT was easily integrated into the GHOST Application Programming Interface. FGB can therefore be used as a tool to create haptic interfaces for GHOST based applications.

IV. Conclusions

There are a number of two dimensional graphical interface builders such as Microsoft's Visual C++ software. These tools are comparable to FGB in some ways, in that they represent an intuitive way to create interfaces that can be viewed as they are created. There are a number of differences, however. Primarily, FGB was built around haptics. Haptics is a tightly coupled component in FGB, which is an important aspect of any haptics software as it is often difficult to simply integrate haptics into applications that were not specifically designed for the need of a high speed (approximately 1000 Hz) interaction loop. Additionally, FGB is entirely three-dimensional and allows direct 3D manipulation. This is essential to both creating and using 3D interfaces where the mouse is often ineffective. A 3D interface is useful to prevent the need for alternating between using the mouse and using the haptic device when working with a 3D application.

As an initial note on the successful use of FGB, it was used to help create itself as it was built. Intuitively, it was expected that an interface such as FGB would be as useful for 3D, haptics based applications as comparable 2D graphical interface builders are for 2D interfaces. It was unexpected, however, that it would be as useful as it was in its own creation. As added functionality was incorporated into FGB, it was easier to create other new control modules that added further functionality. As a finished product, FGB could recreate itself in less than a quarter of the time and effort.

Acknowledgments

The authors would like to thank Brian Pino, Kendall Mauldin, Kevin Oishi, and Dan Zimmerer for their help in the work that lead to this paper. The inclusion of any external products described in this paper does not imply any endorsement by Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the US Department of Energy under Contract DE-AC04-94AL85000.

References

[1] J. Preece, *Human-Computer Interaction*, Addison-Wesley Publishing, New York, NY 1994.
[2] T. Anderson, "FLIGHT Users and Programmers Manual", Sandia National Laboratories, 1999.
[3] W. Barfield, T. Furness III, *Virtual Environments and Advanced Interface Design*, Oxford University Press, New York, 1995.